Naval Research Laboratory

Washington, DC 20375-5320

Automatic Data Sorting Using Neural Network Techniques

BEHROOZ KAMGAR-PARSI

Naval Center for Applied Research in Artificial Intelligence
Information Technology Division

BEHZAD KAMGAR-PARSI

Advanced Information Technology Branch
Information Technology Division

JOHN C. SCIORTINO, JR.

EW Support Measures Branch
Tactical Electronic Warfare Division

February 8, 1996

19960226 117

DTIC QUALITY INSPECTED 1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE February 8, 1996	3. REPORT TYPE AND DATES COVERED Final Report	

4. TITLE AND SUBTITLE

Automatic Data Sorting Using Neural Network Techniques

5. FUNDING NUMBERS

PE - 62270N

6. AUTHOR(S)

Behrooz Kamgar-Parsi, Behzad Kamgar-Parsi, and John C. Sciortino, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/5720--96-9803

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

When several data sources are sending asynchronously without any multiplexing conventions, the stream of data from each source will be interleaved in an unpredictable sequence. In such a sequence, it would be highly desirable to deinterleave the data streams before attempting further processing. After the application of certain signal processing techniques on the incoming interleaved data stream, one obtains a feature space in which different data sources typically form distinct clusters. It is therefore essential to have a reliable clustering technique to determine: (1) the correct number of sources, and (2) the correct membership for each datum. The Hopfield-Kamgar neural net clustering technique appears to be the clustering technique of choice for this task. We explain the main aspects of our technique and briefly discuss alternative neural nets and conventional methods for clustering, in particular as applied to data deinterleaving.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Artificial neural networks Hopfield neural networks	Deinterleaving Algorithms	17
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

AUTOMATIC DATA SORTING USING NEURAL NETWORK TECHNIQUES

1. INTRODUCTION AND BACKGROUND

A major signal processing problem is to correctly separate or deinterleave data streams produced by multiple data sources from the electromagnetic environment. Through signal analysis of data parameters, it is usually possible to deinterleave these streams and assign them to specific sources. This categorization is not always simple, nor is there a unique, provable solution to every categorization problem. It is sometimes necessary to look at inter-relationships of multiple parameters. Often this type of analysis is done manually by experienced analysts. Being able to quickly and accurately identify an incoming data stream automatically and autonomously without operator intervention is quite significant.

This report addresses the data deinterleaving capability of several algorithms that use data feature parameters. Special equipment in the form of feature extractor processors are required to provide feature parameters in real time. For each datum, each feature parameter can be made a component of a vector in feature space. The space this vector occupies can be used to uniquely associate a datum with a source, thereby deinterleaving it from within a group of collected data points. The data points associated with each source tend to cluster together, leaving each cluster associated with a different source.

The feature vector is a high-dimensional quantity that contains a large capability to separate data streams. In this report, only two of the lowest order components are used. Another aspect of the problem is that no initial knowledge of the true number of classes or data sources exists for any given data set. With these limitations, this problem is not uniquely solvable under the standard means of cluster analysis. A self-adapting distribution sampling or clustering algorithm is needed to solve this problem. Based upon preliminary investigations, the use of artificial neural network (ANN) algorithms can provide the necessary capability to solve such a classification problem. The algorithm should use unsupervised learning and be data-shape independent since there is no guarantee that data will cluster in particular shapes.

The algorithm should analyze the data environment by detecting data clusters and determining the parametric limits of those clusters. The data used for this report were collected during field exercises. As real-world data, these collections contain the full spectrum of potential collection distortions. This will allow the developed algorithms to be more robust and adaptable when deployed.

Clustering is an important feature used in many fields, hence, many clustering techniques have been developed. In recent years, clustering techniques inspired by neural networks have been added to the designer's repertoire. Kohonen's [1] self-organizing maps have been used to partition the feature space into distinct regions. Kamgar-Parsi et al. formulated the problem as an optimization problem using the Hopfield [2] network and applied it to compact clustering [3] as well as shape-specific clustering [4]. For simplicity, from now on we will refer to the latter neural net clustering technique as the Hopfield-Kamgar method.

By clustering, we mean partitioning a set of N patterns (the patterns are represented as points in a d-dimensional metric feature space) into K clusters in such a way that those in a given cluster are more similar, in certain ways, to each other than the rest. As there are approximately $K^N K!$ possible ways of partitioning the patterns among K clusters [5], the problem has exponential complexity and finding the best solution is beyond exhaustive search. Yet, neural net techniques are effective in finding practically the best solution in most cases. For this project, we experimented with both the Kohonen and the Hopfield-Kamgar techniques. Since iterative conventional techniques are the most popular clustering techniques, we also compare their performances with the Hopfield-Kamgar technique.

Section 2 of this report describes the Hopfield-Kamgar clustering technique. Section 3 compares the Hopfield-Kamgar technique with the conventional technique. Section 4 describes the Kohonen technique. Section 5 discusses the suitability of the Hopfield-Kamgar vs the Kohonen technique for this project. Section 6 describes the scheme that we have developed for finding the correct number of data sources. Section 7 discusses major new clustering ideas that were developed while working on this project — these ideas greatly enhanced the efficacy and the reliability of the Hopfield-Kamgar clustering technique. Finally, Section 8 contains some remarks about the software and a couple of examples.

2. HOPFIELD-KAMGAR TECHNIQUE

We used analog neural nets for this problem because they outperform digital nets in solving optimization problems [2- 4]. Many problems of interest, including the problem we address in this report, can be cast in terms of an energy function E that is quadratic in the neuronal activities and has the form [2]

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}T_{ij}V_iV_j - \sum_{i=1}^{n}I_iV_i + \frac{1}{\tau}\sum_{i=1}^{n}\int^{V_i}dx\, g^{-1}(x). \tag{1}$$

Here n is the number of neurons in the network, and V_i $(0\le V_i \le 1)$ is the activity or firing rate of neuron i. The first term in Eq. (1) is the interaction energy among neurons. In the second term, I_i is the bias or activity threshold of neuron i. The third term encourages the network to operate in the interior of the n dimensional unit cube $\{0\le V_i \le 1\}$ that forms the state space of the system. In this term, τ is the self-decay time of the neurons, and g, a sigmoid function, is the gain or transfer function of the neurons that relates the input u_i to the output V_i. A standard form for g, which we will also use, is

$V_i = g(u_i) = \frac{1}{2}(1+\tanh\frac{u_i}{u_0}) = \frac{1}{1+\exp(-2u_i/u_0)}$, where u_0 determines the steepness of gain. The

neuronal activities V_i as well as the input signals u_i depend on time t. The evolution of the network is determined by the n coupled ordinary differential equations $du_i/dt = -\partial E/\partial V$, which are

$$\frac{du_i}{dt} = -\frac{u_i}{\tau} + \sum_{j=1}^{n}T_{ij}V_j + I_i. \tag{2}$$

We will set $\tau = 1$, so that time is measured in units of τ. To find a solution (i.e., a minimum), we start the network from a randomly selected state and let it evolve until it reaches a minimum of the function and stops [6].

We want to partition a set of N points in a 2-D plane (generalization to arbitrary dimensions is trivial) into the *best* K clusters — best in the sense that the sum of the squares of the distances of the points from their respective cluster centroids (i.e., sum of "within cluster variances") is minimized. We formulate the problem in a manner that can be solved by a neural network; that is, we cast the problem in terms of an energy function that can be minimized by the network.

The energy function will consist of two parts: (i) constraint terms which make certain a point, at the end of the search, belongs to one and only one cluster; (ii) the cost term which is the sum of the residuals and is the function we actually wish to minimize. To partition N points (patterns) among K clusters, we need to have $n = K \times N$ neurons, i.e., K neurons for each point. We denote each neuron by two letters, e.g., neuron pi, and let V_{pi} stands for its activity. The magnitude of V_{pi} represents the weight with which point i belongs to cluster p. The energy function E can then be expressed as

$$E = \frac{A}{2} \sum_{i=1}^{N} \sum_{p=1}^{K} \sum_{q \neq p}^{K} V_{pi} V_{qi} + \frac{B}{2} \sum_{i=1}^{N} \left(\sum_{p=1}^{K} V_{pi} - 1 \right)^2 + C \sum_{p=1}^{K} \sum_{i=1}^{N} R_{pi} V_{pi}^2,$$ (3)

where the coefficients A, B, and C are positive constants. The A-term and the B-term are the constraint terms, and together they enforce the syntax of the problem. The C-term is the cost term, where R_{pi}, the square of the distance of point i from the centroid of cluster p (i.e., the residual), is given by $R_{pi} = \left(x_i - X_p \right)^2 + \left(y_i - Y_p \right)^2$ are the coordinates of point i, and $\left(X_p, Y_p \right)$ are the coordinates of the centroid of cluster p. The centroid of cluster p is obtained from the weighted average shown in Eq. (4). Generalizing to higher dimensions is straightforward.

$$X_p = \sum_{i=1}^{N} x_i V_{pi} \bigg/ \sum_{i=1}^{N} V_{pi}$$

$$Y_p = \sum_{i=1}^{N} y_i V_{pi} \bigg/ \sum_{i=1}^{N} V_{pi}.$$ (4)

The network dynamics, obtained from $-\partial E / \partial V_{pi}$, are

$$\frac{du_{pi}}{dt} = -u_{pi} - A \sum_{q \neq p}^{K} V_{qi} - B \left(\sum_{q=1}^{K} V_{qi} - 1 \right) - C R_{pi} V_{pi} + I_{pi}.$$ (5)

To find a solution, we assign random values between 0 and 1 to all the $n = K \times N$ neuronal activities, V_{pi}, and solve n ordinary differential equations until the activities of all the neurons cease to change.

For simulations, we have chosen the following values for the parameters of the energy function: $A = B = 1$, $C = 0.9/R_{avg}$, all $I_{pi} = 1$, and the gain function parameter $u_o = 0.1$. Scaling parameter C with the average residual R_{avg} is necessary to ensure good solutions, because as the network evolves, the residuals become generally smaller and the cost term becomes less effective in driving the network toward good solutions. This rescaling of parameter C keeps the cost term of the same order of magnitude as the syntax terms. The parameter values are selected by analyzing dynamical stability of valid solutions [7]. Also, for a general discussion of the effectiveness of the Hopfield network, the reader is referred to Ref. 8.

3. COMPARISON OF CONVENTIONAL AND HOPFIELD-KAMGAR TECHNIQUES

To compare the performance of the Hopfield-Kamgar neural net technique with a popular conventional technique, we have experimented with some data sets. In the first data set, there are 128 points divided among five clusters with within-cluster Gaussian distributions (Fig. 1(a)). Here the five clusters are rather well defined, and out of the 128 trials, the neural net found the optimum clusters 128 times. The conventional method of Forgy [9] in 128 trials found the best clusters only 46 times and

various other solutions 82 times. Clearly, in this example, the neural net outperforms the conventional method, in that it finds the best solution much more frequently.

To test the performance of the network in cases where the clusters are fuzzy, we started from the data points of Fig.1(a), randomly selected 10% of the points, and distributed them uniformly throughout the unit square (Fig.1(b)). Thus we obtained 5 clusters with uniform background noise. The neural net in 128 trials found the best clusters 28 times. It failed to find valid solutions satisfying the syntax 46 times. This large number of failed solutions can be interpreted as an indication that the clusters are fuzzy, that there are outliers, and that perhaps the specified number of clusters, $K = 5$, is too few. However, even when the syntax is not satisfied, we can extract a valid solution with the following scheme. For each point i, set the largest V_{pi} to 1 and all the other V_{qi} with $q \neq p$ to 0, and interpret this solution as the one favored by the network; thus, we obtain 128 solutions. Conventional algorithms always find valid solutions and cannot give an objective indication of the fuzziness of clusters.

Similarly to Fig. 1(b), we generated other data sets by increasing the background noise to 25%, 50%, 75%, and 100% (i.e., no clusters). These data are shown in Fig. 1(c-f). Table 1 lists the results of 128 trials with partitioning the data among five clusters using the neural net and Forgy's method. The average estimated convergence times for the network are given in units of τ. Two points should be noted in this table:

1. As the five clusters become less discernible, the network increasingly fails to satisfy the syntax, indicating that the clusters are fuzzy and that five clusters are not sufficient. The conventional method, on the other hand, always finds valid solutions. Although the variety of solutions that it finds increases (this is true in both methods), it may be taken as a clue to the fuzziness of the clusters. It is not as objective an indicator as the failure to satisfy the syntax.

2. When there are well-defined clusters, the neural net performs better than the conventional techniques, which is reflected in the lower average (is the sum of within-cluster variances) for the solutions found by the neural net. As the clusters become fuzzier, the quality of the solutions found by both methods becomes comparable.

For the data of Fig. 1(f), the centroid trajectories converge to many different points for different trials, which shows that where there is no underlying structure in the data, the network does not prefer any particular clustering and, hence, finds many different solutions.

To test how this neural net algorithm scales with the size of the problem, we generated 32, 64, 128, and 256 data points, distributed among five rather well-separated clusters. In all the cases, the network found valid solutions in all of its trials, and the best solution in nearly all of its trials, consistently performing better than Forgy's algorithm. Thus, it appears that the Hopfield-Kamgar clustering technique scales well.

We briefly mention that in another example we used a standard clustering data set, namely Anderson's four-dimensional iris data (Fig. 2), which is composed of three well-separated clusters. As was the case with the example of Fig. 1(a), the neural net approach found the best solution in all of its 100 trials, whereas the conventional technique found the best solution in 78% of its trials.
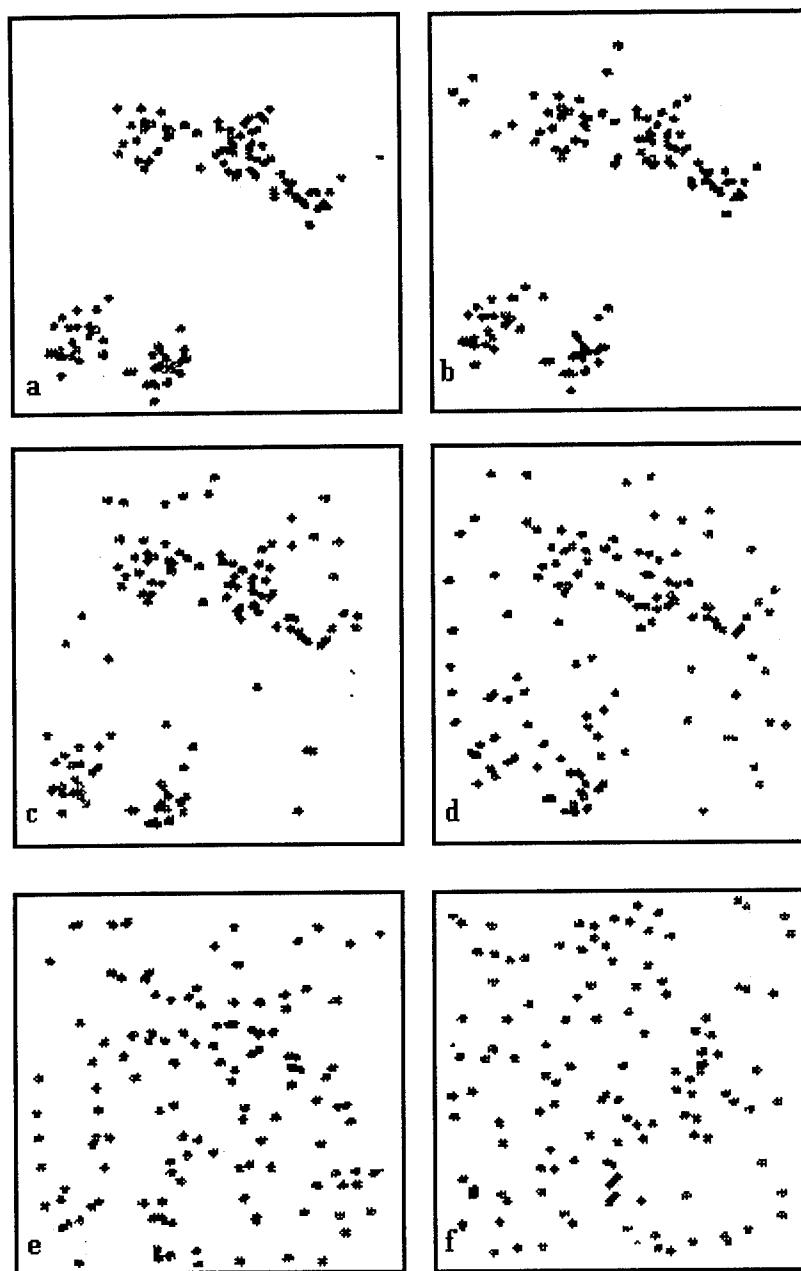
Fig. 1 — 128 data points in the unit square: (a) 5 well-defined clusters with within-cluster Gaussian distributions; (b) same clusters with 10% uniform background noise; (c) with 25% noise; (d) with 50% noise; (e) with 75% noise; and (f) uniform distribution or 100% noise.
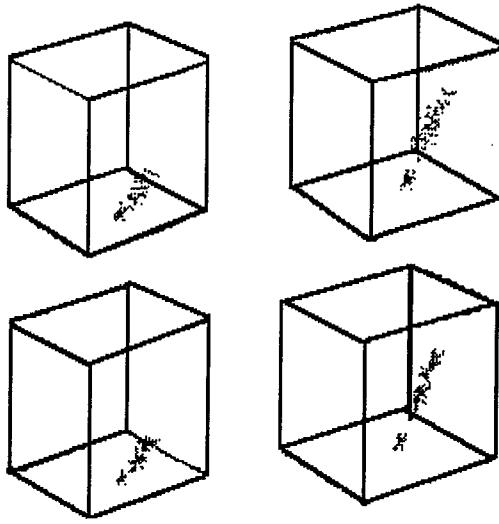
Fig. 2 — Anderson's four-dimensional data set composed of 150 samples of three species of iris: (upper) two different three-dimensional profiles of Anderson's data set, and (lower) two cluster representations of the profiles immediately above them.

Table 1—Forgy's Conventional Algorithm Results Compared with those Obtained by the Neural Network (Based on 128 Trials)

Data	Conventional			
	Iter	Best Var	Best %	Avg Var
a	7	0.62	36	1.23
b	8	1.06	34	1.57
c	8	1.95	12	2.27
d	10	2.94	2	3.14
e	10	3.88	10	4.11
	10	4.46	2	4.64

Data	Neural Net				
	Time	Best Var	Best %	Avg Var	Synt %
a	4	0.62	100	0.62	100
b	7	1.06	22	1.24	64
c	7	1.95	19	2.03	9
d	8	3.00	15	3.04	0
e	6	3.89	1	4.11	1
f	8	4.46	2	4.71	0

Note: The following items related to Table 1 are referenced to the data in Fig. 1(a-f). *Data* refers to the data points of Fig. 1(a-f). *Iter* is the average number of iterations for convergence. *Best Var* is the variance of the best solution found. *Best%* is the percentage of trials that found the best solution. *Avg Var* is the average variance of the solutions found. *Time* is the average estimated time of convergence in units of τ. *Synt%* is the percentage of trials that found solutions satisfying the syntax.

4. THE KOHONEN TECHNIQUE

A technique that may be used for unsupervised clustering of data is based on Kohonen's Self-Organizing Map principle [1]. The idea behind this technique, often called competitive learning, is as follows: Assume a sequence of (statistical) samples of a vectorial observable $\mathbf{x} = \mathbf{x}(t)$, where t is the time coordinate, and a set of variable reference vectors $\mathbf{y}_i(t)$, $i = 1, \cdots, K$. Both $\mathbf{x}(t)$ and $\mathbf{y}_i(t)$ are D-dimensional vectors. Assume that $\mathbf{y}_i(0)$ are initialized in some manner. If $\mathbf{x}(t)$ can be compared with each $\mathbf{y}_i(t)$ at each succesive time step, taken to be integer $t = 1, 2, 3, \cdots$, then the best matching $\mathbf{y}_i(t)$ is to be updated to match even more closely the current $\mathbf{x}(t)$. This is an approach to the classical Vector Quantization where one wants to produce an approximation to a (continuous) probability density function $p(\mathbf{x})$ of the input vector $\mathbf{x}$ using a finite number of codebook vectors $y_i, i = 1, \cdots, K$.

For unsupervised clustering, the Self-Organizing Map may be implemented in various ways. Below we describe two implementations: one-pass and iterative. In this work, the distance metric is the Euclidean distance, and the algorithms are based on minimizing the square-error criterion.

4.1 One-Pass Implementation

The initial approach taken is the following. First, normalize all the coordinates to the range $(-1, +1)$, i.e, the feature space is a D-dimensional hypercube. Select a number of codebooks larger than the number of expected data sources. Initialize the codebooks by placing them randomly near the center of the hypercube, $\mathbf{y}_i(0) \approx 0$. Then process the incoming data points represented by the feature vector $\mathbf{x}(t)$, $t = 0, \cdots, N-1$, where N-1 is the number of data points. For data point t, find the codebook vector $\mathbf{y}_i(t)$ closest to $\mathbf{x}(t)$, i.e.,

$$\left\| \mathbf{x}(t) - \mathbf{y}_i(t) \right\| = \min_k \left\| \mathbf{x}(t) - \mathbf{y}_i(t) \right\|, \tag{6}$$

and update that codebook vector according to

$$\mathbf{y}_i(t+1) = \mathbf{y}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{y}_i(t)], \tag{7}$$

while leaving other codebooks unchanged. Here $\alpha(t)$ is a monotonically decreasing sequence of scalar-valued gain coefficients, $0 < \alpha(t) < 1$. The usual choice for $\alpha(t)$ is

$$\alpha(t) = \alpha_0 \left(1 - \frac{t}{t_m} \right), \tag{8}$$

where $0.1 < \alpha(t) < 0.9$ and $t_m \geq N$. Each time a codebook is updated, its *counter* is incremented by one to keep track of the number of data points assigned to it. The codebook counter may be used to infer the number of clusters (data sources); if its value is high, the codebook is considered to be a legitimate cluster, otherwise the codebook is disregarded.

This implementation is very fast. However, it is sensitive to the order in which the data is processed and also to the initialization of the codebook vectors. It can also be sensitive to the threshold used for deciding whether a codebook represents a sufficient number of data points.

4.2 Iterative Implementation

We have experimented with an iterative implementation of the Kohonen technique which is as follows. We label the points not by time but by an index, say $j = 1, \cdots, N$. Thus the data are represented by feature vectors $\mathbf{x}_j$. First the codebook vectors (or cluster centroids) are initialized randomly. Then the centroids are updated by presenting the data to them repeatedly over many iterations. At iteration t the $\mathbf{y}_i(t)$ are updated as follows: find the centroid closest to the data point j and move that centroid toward this point. That is,

$$\mathbf{y}_i(t+1) = \mathbf{y}_i(t) + \alpha(t) M_{ij}(t)\big[\mathbf{x}_j - \mathbf{y}_i(t)\big],\tag{9}$$

where

$$M_{ij} = \begin{cases} 1 & \text{if } \left\|\mathbf{x}_j - \mathbf{y}_i(t)\right\| = \min_k \left\|\mathbf{x}_j - \mathbf{y}_k(t)\right\| \\ 0 & \text{otherwise} \end{cases}.\tag{10}$$

Here, M_{ij} are the elements of the cluster membership matrix, i.e., if $M_{ij} = 1$, then point j belongs to cluster i. Note that during one iteration all the data points are presented, while coefficient $\alpha(t)$ is held fixed. As the iterations proceed, $\alpha(t)$ decreases monotonically according to the usual recipe given above, where now $500 < t_m < 10{,}000$ depending on the desired accuracy. As an empirical rule of thumb, it is suggested that $t_m \approx 500K$, where K is the number of codebooks [10].

The iterative implementation is obviously slower because the basic one-pass computations are repeated many (typically hundreds of) times. Its advantage is that the outcome is far less sensitive to the order of data presentation and possibly less sensitive to initial centroid locations. Furthermore, the codebook vectors $\mathbf{y}_i$ are likely to be the correct centroids of the clusters. The number of points belonging to cluster i, or the codebook counter, which may be used to infer the number of clusters, is $N_i = \sum_j M_{ij}$.

5. RELATIVE MERITS OF HOPFIELD-KAMGAR AND KOHONEN METHODS

As explained in Section 3, the Hopfield-Kamgar technique is more dependable than the conventional techniques. The question is, therefore, whether to employ the Kohonen method or the Hopfield-Kamgar method. After a careful examination of the advantages and disadvantages of the two methods, we concluded that the Hopfield-Kamgar technique appears more suitable for the project at hand. The main considerations leading to this conclusion are as follows.

- Accuracy of the Solutions: While the Hopfield-Kamgar method gives accurate cluster centroids, there is no theoretical reason why the Kohonen method should produce precise centroid locations—unless the number of iterations is exceedingly large.

- Correctness of the Final Number of Clusters: Employing the Kohonen technique, the final number of data sources must be decided by comparing the numbers of data points within different clusters. Those codebooks (clusters) that contain relatively few points are not considered to be true sources of data. In the Hopfield-Kamgar technique, this matter is decided by a more objective and elaborate scheme (as described in the following section).

- Accuracy of Effective Initial Guess for Number of Clusters: Using the Kohonen method, the starting cluster number cannot be smaller than the actual cluster number. Therefore, to be on the

safe side, the user would have to supply the program with a starting cluster number that is considerably larger than what the user actually thinks it is. In a way, the effective initial guess should reflect the worst case scenario. In the Hopfield-Kamgar method, the initial guess can be smaller (or larger) than the actual number of data sources. This allows the user to supply the program with his best guess.

- Computational Costs: In the Hopfield-Kamgar technique, computations are more involved than those in the Kohonen technique. If high precision is not needed for the centroid locations, then the Kohonen technique is likely to be more efficient. However, two factors, namely insistence in great precision and poor initial guess for cluster numbers, could lead to comparable efficiency for the two techniques.

- Potential for Extension to Complex Clusters: The Hopfield-Kamgar method has already been used to solve shape-specific clustering [4] as well as compact clustering, and has potential to be extended to elongated clustering and other non-compact clusterings. It is not clear how the Kohonen method could be extended to deal with non-compact clusters.

6. FINDING THE CORRECT NUMBER OF CLUSTERS

One should distinguish between the following two problems:

1. Given a set of patterns, partition it into M clusters so that patterns within each cluster are more similar to each other than the rest, where M is a prespecified number,

2. Given a set of patterns, determine the correct number of clusters that exist in it.

These two problems are generally quite different and thus they require different solutions. Most of the clustering algorithms assume that the number of clusters is already known, and require the user to supply that information. However, for the problem of multi-source data deinterleaving, the true number of clusters is generally unknown. Therefore, we need to solve both of the above problems simultaneously. The scheme that we have developed for determining the number of clusters is described below.

The user will supply an initial estimate of the number of clusters, say, k. The program will then partition the data points into k clusters using the technique described earlier. Next, for each of the k clusters, it will calculate its variance, σ_i^2, for the ith cluster. For each pair of clusters, the two standard deviations, σ_i's are then added up and normalized with respect to the distance between their respective centroids. Now, consider a pair of clusters, say, i and j, and define their "pairwise normalized s.d.," $\tilde{\sigma}_{ij} = (\sigma_i + \sigma_j)/d_{ij}$, where d_{ij} is the distance between the centroids of i and j. If $\tilde{\sigma}_{ij}$ is a small number, then the two clusters must be well separated, but if it is large, they may be too close to each other to be considered as two separate clusters. To implement this scheme reliably, one must quantify the notion of large or small values for pairwise normalized standard deviations.

Here we assume that within-cluster variations follow normal distributions. That is, the variations of data belonging to the same cluster follow a Gaussian distribution around the centroid of that cluster. Note that because the main source of noise is jitter induced by the source, such an assumption is valid in most cases. Thus, to decide whether the two clusters i and j should be considered as two or one, we will calculate the overlap between the two Gaussians that are defined by the data points assigned to the two clusters. The center of a Gaussian is assumed to be the centroid of that cluster, and its width is taken to be the variance within that cluster. Recall that in a Gaussian distribution, the population confined within two standard deviations of the mean comprises 95% of the total population. Therefore, if $\tilde{\sigma}_{ij} < 1/2$, then

clusters i and j can be regarded as two separate clusters with great confidence. On the other hand, if $\tilde{\sigma}_{ij}$ >1, we can confidently assume that i and j are not two separate clusters. This is a conservative bound because for identical Gaussians, if $\tilde{\sigma}_{ij}$ >0.9, then the two cannot be resolved. If neither of the above conditions is satisfied, the two·clusters may or may not be distinct. In this case, we need to develop a more elaborate test to decide the matter.

Two approaches come to mind: (i) Determine whether the combined data in the two subpartitions agrees with the definition of a single Gaussian or not. (ii) Assume that there are two Gaussians and determine whether the two can be resolved or not. In the first approach, the centroid and the standard deviation of the combined data are first calculated. Next, data within certain radii, which are certain fractions of the standard deviation from the centroid, are counted to determine whether they are in agreement with the definition of a Gaussian distribution or not. If the number of data points within a given radius is not enough, then the two subpartitions cannot be regarded as one, as there is an empty or sparse space between them. The second approach is discussed below.

Consider the following pairwise density function for clusters i and j,

$$f_{ij}(x) = \frac{N_i}{\left(2\pi\sigma_i^2\right)^{d/2}} \exp\left[-(x-y_i)^2\big/2\sigma_i^2\right] + \frac{N_j}{\left(2\pi\sigma_j^2\right)^{d/2}} \exp\left[-(x-y_j)^2\big/2\sigma_j^2\right], \tag{11}$$

where d is the dimension of the feature space, N_i is the number of data points belonging to cluster i, and y_i is its centroid. This density function has either one or two peaks. If $f_{ij}(x)$ has two peaks, then the pairwise distribution is bimodal and the two clusters are, indeed, two. If there is only one peak, then the two clusters overlap substantially, that is, they are not distinct and should be merged. Because of the spherical symmetry of the Gaussian distributions, the peaks lie between the two centroids on the line connecting y_i to y_j. Thus, the problem of finding the peaks is a simple one-dimensional search on the line connecting the centroids. We note that this second approach is appropriate for supervised clustering where the data points are labeled *a priori*.

We further note that in this cluster validation step, we have assumed that within-cluster variations follow normal distributions. We have further assumed that within-cluster variations are identical for all features (or variables), hence a spherically symmetric Gaussian distribution. A second assumption is that the data sources are (roughly) equally active, that is, the number of data points in each cluster is comparable, $N_i \approx N_j$. Otherwise, genuine, but weak, clusters may be smeared by stronger ones. These assumptions in the cluster validation step agree with those in the clustering technique, which is based on the assumption that clusters are basically round, compact, and of roughly equal population. For clusters that do not follow these assumptions, i.e., if they are elongated or have considerably different populations, one must develop new techniques. The neural net technique we have used in this work can potentially be generalized to handle these situations.

If we construct the pairwise density function with the assumption that a single source distribution can have the general form

$$\frac{N_i}{(2\pi)^{d/2}\left|\sum_i\right|^{1/2}} \exp\left[-\frac{1}{2}(x-y_i)^T \sum_i^{-1}(x-y_i)\right], \tag{12}$$

where Σ_i is the covariance matrix, and the Euclidean distance is replaced by the Mahalanobis distance, then the peaks of $f_{ij}(\mathbf{x})$ would no longer lie on the line connecting the centroids, hence finding the peaks turns into a difficult optimization problem.

7. OTHER NEW CLUSTERING FEATURES

To increase the reliability and efficiency of our technique, many new ideas were developed. These new ideas were tested on a sample of real-world data, and the useful ones were incorporated in the final software. These new features are described in the following paragraphs.

7.1 Two-Stage Solution

To partition N data points into M clusters, one must use a network that contains $M \times N$ neurons. When the number of neurons is large and the initial guess is poor (both in terms of their locations and their numbers), then not only each iteration could take a long time, but also the number of iterations required for convergence may be too large.

The initial activity of each neuron is decided by a random number assigned to it. These initial activity rates imply certain initial locations for the centroids of the clusters. If the initial locations are poor, then it may take many iterations to converge (if it does at all). Furthermore, if the number of neurons is large, each iteration could take a long time, computational complexity per iteration is $\mathcal{O}(MN)$. We have found that the following approach alleviates this problem. For a given value of M, the number of clusters, first we solve (or attempt to solve) the problem for only a small random sample of data points, say $N_s \approx 0.05N$, thus cost per iteration is reduced by a factor of 20. If this problem is solved, then the final locations of the centroids will be used as the initial locations of the centroids for the entire data set; that is, the initial activity rate of each neuron is set to 1 if this neuron associates the data point with its nearest centroid, and to 0 otherwise. Because these initial firing rates are usually very good, the network converges in a few iterations.

7.2 Handling Poor Initial Values for Cluster Numbers

Our experience shows that for the case where the initial guess for the number of clusters is larger than the correct number, then the syntax is rarely satisfied, at least not for any reasonable upperbound on the number of allowed iterations. Therefore, we have introduced a fairly tight upperbound for the number of iterations allowed for convergence for a given number of clusters M, when only a fraction of data points is used. If this upperbound is exceeded, the program will restart not only from a new set of initial firing rates for the neurons, but also with a reduced number of clusters, i.e., M will be reduced by 1. Since this upperbound is tight, it is possible, though not common, that when M is reduced to its correct value, the upperbound is exceeded before convergence. However, this is not a serious drawback, because the program will begin to increase M once convergence has been achieved for a certain value of M.

7.3 Merging or Splitting and Activities of Neurons

If the two clusters i and j are found not to be distinct when using the criteria described in the previous section, then the two are simply merged as one cluster. Thus, M is reduced by one and the program restarts, but the initial activities of the neurons will not be set to random numbers, instead they are set such that the starting centroids will be as follows. The centroid of the newly formed cluster is taken to be the (weighted) mid point between the centroids of subclusters i and j, and the centroids of other clusters will be what they were.

When the clustering is complete for a particular value of the cluster numbers M, and all the clusters are found to be distinct, then M is increased by one and the program restarts. However, the initial activities of neurons will not be set to random numbers; instead, they are decided by the initial centroids. That is, the initial activity of a neuron is set to 1 if that neuron associates the data point with its nearest centroid, and to 0 otherwise. To determine the locations of the initial centroids, the following test is done. One by one, each of the original M clusters is partitioned into two subclusters, and the one splitting into two most distinct clusters is picked to contribute two initial centroids. That is, the two centroids resulting from the split of this cluster together with the centroids of those clusters which were not split will serve as the initial centroids when the program begins to partition the whole data set into $M+1$ clusters. It is noted that the partitioning of each cluster into two subclusters is also done in two stages according to Section 6.

8. THE SOFTWARE AND SOME EXAMPLES

The original neural net clustering program was written in Fortran. For this project, the program was translated to C and many new features were added to it. The software now contains about 700 lines of code and includes about 18 functions. It uses Pixrects to display two-dimensional results. To have the required flexibilities, the program uses mostly dynamic memory allocation. To run the program, the command line must have the following format

compiled-name $M\ N\ C$ <input-file >output-file,

where M is the initial guess for the number of clusters and N is the number of data points. C is a code to invoke Pixrect to display the data and the calculated cluster centroids if so wished: if the display is desired, C must be set to 20, otherwise there will be no display. The software was tested on many examples; below, we present two cases. Figure 3(a) displays a data set consisting of 415 data points coming from a single data source. The program was able to correctly detect the presence of only one data source. The dark square shows the calculated cluster centroid. Figure 3(b) displays a data set consisting of 896 data points coming from three data sources. The program was able to find the correct solution (the three dark squares) from any initial guess for the number of data sources or any set of initial firing rates for the neurons.
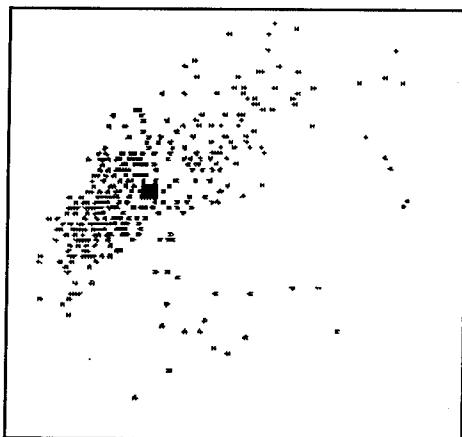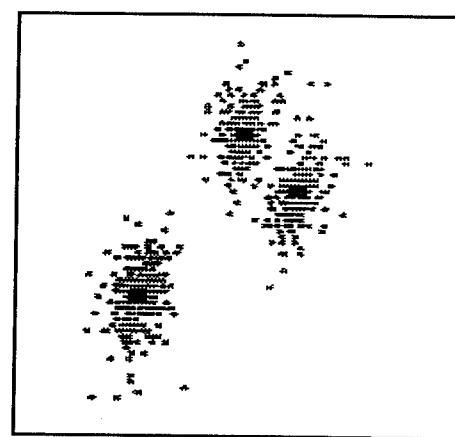
a   b

Fig. 3 — (a) The program detects the presence of only one data source. (b) The program finds the correct solution (dark squares as centroids) from every initial guess for the neuronal activity rates or number of data sources.

9. CONCLUSIONS

The results reported here have made a significant positive impact on accurately partitioning data and revealed insight for further research into applying methods of artificial intelligence in signal processing. Both the Hopfield-Kamgar and LVQ neural networks were examined with the primary emphasis on the former technique.

The Hopfield-Kamgar algorithm takes the data environment and groups the data into bins, groups, or clusters, each of which contains only one data source for further processing. The algorithm must analyze the data environment and detect data clusters. A key aspect of the Hopfield-Kamgar technique is that it creates a high-quality, statistically accurate cluster count within a case. To a very high accuracy, the data are correctly associated with each other into their respective data bins. In each case, it was not known ahead of time how many clusters were present; nor was it known even what upper limit to the number of data sources would be encountered in any given case or collection.

The data analyzed has a time structure based upon two signal characteristics: the data repetition interval and a data frame period. The former is a block of data coming in a specified burst from the source, while the latter determines the time or period between bursts. Even the order of data presentation contains significant infomation about the data sources. Further investigation of the time structure of the data is recommended.

The Hopfield-Kamgar network was implemented with only two feature parameters, and was found to arrive at highly accurate partitions of data into data source bins. The data set was, however, somewhat limited in this initial investigation. Other studies using very large databases with thousands of cases indicate that a minimum of eight feature parameters is needed to deinterleave most data sources from each other. The full feature parametric set is necessary to maximally deinterleave the data.

The algorithm was developed on a Sun Workstation in software. The execution time is good for a software simulation of a neural network. A significant increase in execution speed on the order of 10 to 1000 times could be accomplished by implementing the algorithm in either electrically programmable read only memory (EPROM) chips or application specific integrated circuits (ASICs). This would allow for the full spectrum of high-speed and real-time applications [11].

The algorithm is targeted at four major applications: (1) control of software bin parameter limits to sort data sources, thereby reducing processing burdens, (2) control of hardware bin parameter limits in an advanced signal processor developed under the 6.2 S&T block, (3) as an "analyst's aid" in the manual analysis of data collections, and (4) as an automatic software-driven deinterleaver in the front end of an automatic analysis system.

REFERENCES

1. Kohonen, T. (1989). *Self-Organization and Associative Memory*, 3rd edition, Springer-Verlag.

2. Hopfield, J.J. and Tank, D.W. (1985). "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics* **52**, 141.

3. Kamgar-Parsi, B.; Gualtieri, J. A.; Devaney, J. E.; and Kamgar-Parsi, B. (1990). "Clustering With Neural Networks," *Biological Cybernetics* **63**, 201.

4.   Kamgar-Parsi, B.; Kamgar-Parsi, B.; and Wechsler, H. (1990). "Simultaneous Fitting of Several Planes to Point Sets Using Neural Networks," *Computer Vision, Graphics, and Image Processing* (CVGIP) **52**, 341.

5.   Feller, W. (1959). *An Introduction to Probability Theory and Its Applications*, 2nd edition, Vol. 1, p. 58, John Wiley.

6.   Gear, C.W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall.

7.   Kamgar-Parsi, B. and Kamgar-Parsi, B. (1987). "An Efficient Model of Neural Networks for Optimization," *Proceedings of the IEEE First International Conference on Neural Networks*, Vol. 3, p. 785.

8.   Kamgar-Parsi, B. and Kamgar-Parsi, B. (1992). "Hopfield Model and Optimization Problems," *Neural Networks for Human and Machine Perception*, edited by H Wechsler, Academic Press, Vol. 2, p. 94.

9.   Forgy, E.W. (1965). "Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications," Biometric Soc. Meeting, Riverside, California. *Abstracts in Biometrics* **21**: 768.

10. Kohonen, T. (1990). "The Self-Organizing Map," *Proc. IEEE* **78**, 1464.

11. Mead, C. (1989). *Analog VLSI and Neural Systems*, Addison-Wesley.